UNIVERSIDADE
DE VIGO

ESCOLA SUPERIOR DE ENXEÑARÍA

Memoria do Traballo de Fin de Grao que presenta

D. Julián Jiménez González

para a obtención do Título de Graduado en Enxeñaría Informática

SuperCars: Desarrollo de videojuego para dispositivos móviles

Xaneiro, 2015

Traballo de Fin de Grao Nº: EI 14/15 - 010

Titor/a: Alma María Gómez Rodríguez
Área de coñecemento: Linguaxes e Sistemas Informáticos
Departamento: Informática

Índice

1. Descripción del Proyecto

1.1. Introducción

Desde que se comenzaron a comercializar en la década de los 90, el concepto de "videojuego" no ha hecho más que evolucionar hasta lo que conocemos hoy en día, llegando incluso en algunos países al estatus de medio artístico.
Así ha sido en EEUU, donde el MoMA[1] ya ha introducido un buen número de videojuegos entre sus colecciones y exposiciones (*Video Games: 14 in the Collection, for Starters (Paola Antonelli, MoMA, 2012)*
).

Tanto es así que, según el informe anual de 2014 de NewZoo (*Ilustración 1*), la industria del videojuego generó más de $80 billones y cuenta con más de 1700 millones de jugadores en todo el Mundo.

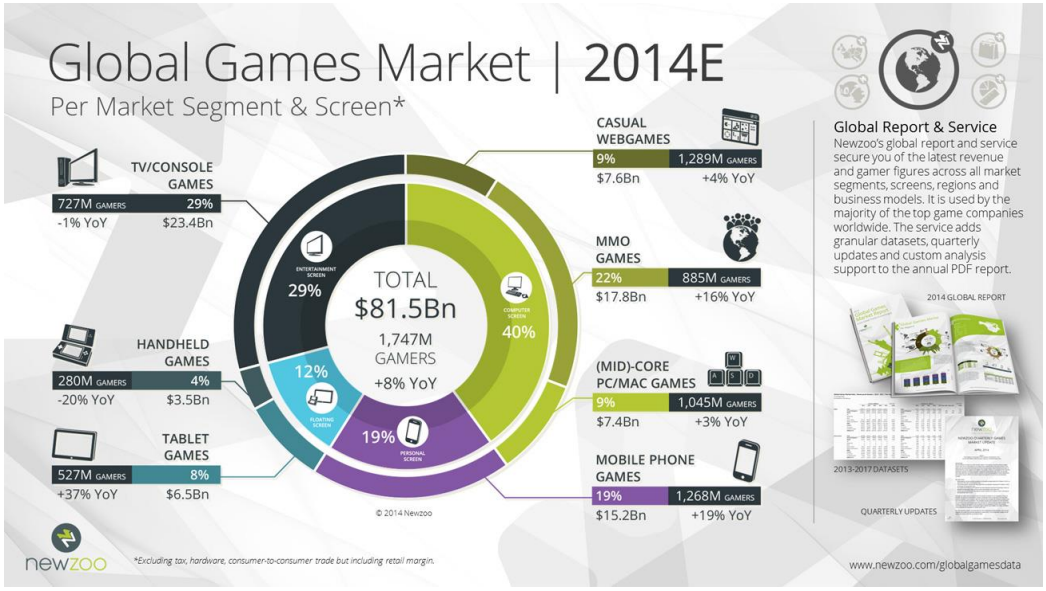


Ilustración 1. *Informe segmentado de NewZoo sobre la industria del videojuego en 2014*

Se considera que en 2015 se podrían alcanzar los $100 billones, poniéndose a la altura de otras industrias de entretenimiento como la música y el cine.

Esta situación favorece la aparición de cada vez más salidas profesionales en el sector, ya que cada vez más disciplinas se ven involucradas en el desarrollo de un videojuego a nivel profesional. Entre otras, podemos encontrar:

- **Desarrollo de Software y herramientas.** Programadores junior/senior, ingenieros de Software, físicos, matemáticos…
- **Sonido.** Ingenieros de sonido, músicos, compositores, cantantes…
- **Arte.** Modeladores 3D, pintores, dibujantes, diseñadores, artistas gráficos, escritores...

[1] *Museum of Modern Art (Nueva York)*

– **Telecomunicaciones.** Ingenieros de Telecomunicaciones, ingenieros en Tecnologías de la Información…

Además, existe mucha documentación y herramientas que permiten a cualquier persona desarrollar su propio videojuego, tanto si ya tiene los conocimientos necesarios como si está interesada en aprenderlos.
Por otra parte, hoy en día hay muchas plataformas de ayuda mutua para desarrolladores de Software en general (como StackExchange o reddit), además de foros específicos y soporte de los propios desarrolladores de cada herramienta.

Entre las distintas plataformas de videojuegos, la de dispositivos móviles (*Smartphones* y *Tablets*) destaca como la de mayor auge en la actualidad, y (como refleja la *Ilustración 1*) la que mayor crecimiento anual prevé.

Por su parte, el SO Android cuenta con una cuota del mercado global de dispositivos móviles de alrededor del 85% (*Ilustración 2*), siendo videojuegos más del 40% de las aplicaciones descargadas para estos dispositivos.

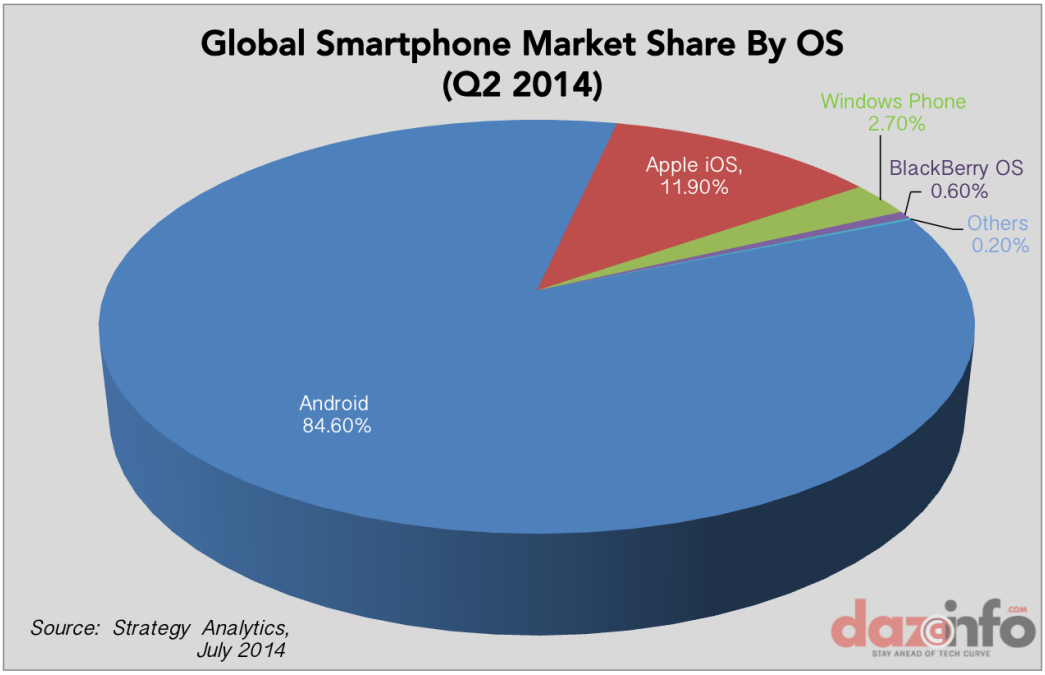


Ilustración 2. *Distribución global de SO para Smartphones 2014 (Dazeinfo)*

Esto puede relacionarse directamente con el creciente interés de las personas por los juegos "casual", a los que no necesitan dedicar mucho tiempo y que les permiten distraerse durante unos minutos de sus actividades cotidianas. Además, el reducido coste de estos videojuegos para plataformas móviles los hace mucho más accesibles para todo el mundo.

Es por todo ello que se ha decidido realizar un videojuego para Android para este TFG[2].

[2] *Trabajo de Fin de Grado*

1.2. Objetivos

El objetivo de este proyecto es desarrollar un videojuego de carreras en 2D de un jugador para Android.

Desde los menús se podrán realizar las configuraciones mínimas de la carrera y competidores (número de vueltas, dificultad, número de contrincantes, etc…).

Durante la carrera se tendrá una vista aérea y se podrán ver los contadores de tiempo, vueltas y posición en carrera. Los rivales tendrán distintos niveles de dificultad. Esto influirá en los parámetros de su vehículo o su comportamiento durante la carrera.

Además de la línea del circuito, habrá zonas que modifiquen el comportamiento del vehículo (como arena o hierba) y muros que lo limiten.

También se guardará un ranking de vueltas rápidas.

1.3. Documentación entregada

La documentación correspondiente a este TFG es la siguiente:

1.3.1. Memoria de Trabajo

Este mismo documento, en el cual se recoge toda la información referente a la planificación y gestión del proyecto, así como los desvíos producidos en tiempo y forma con respecto a dicha planificación.

1.3.2. Manual de Usuario

Documento en el que se describe de forma visual cómo funciona la aplicación de cara a los usuarios finales de la misma.

1.3.3. Manual Técnico

En este documento se detalla la aplicación desde el punto de vista de un desarrollador, facilitando las tareas de mantenimiento y ampliación de la misma.

Nótese que toda la documentación de diseño de la aplicación se presenta sólo en su versión final, a pesar de que se planteó al inicio del proyecto que se detallaran las distintas fases por las que ha pasado esta documentación.
Esto se debe a que se considera que buena parte de la información sería redundante debido a que los cambios entre una fase y otra no tienen por qué ser sustanciales.

1.3.4. Código fuente

En un CD se adjuntará el código fuente de la aplicación con la estructura siguiente:

- **/codigo**. Directorio que contendrá los archivos del código fuente.
- **SuperCars.apk**. Archivo de instalación de la aplicación. Para instalar la aplicación desde este archivo es necesario permitir en el

dispositivo Android la instalación de aplicaciones de orígenes desconocidos[3].

2. Proceso de desarrollo, tecnologías y herramientas

Teniendo en cuenta la inexperiencia del desarrollador con el entorno de desarrollo y las tecnologías utilizadas para este proyecto, se ha considerado que un desarrollo ágil (inspirado en SCRUM) es lo más adecuado, permitiendo adaptarse a posibles cambios o problemas que surjan durante el proyecto.

En ningún caso se hace referencia a SCRUM como proceso de desarrollo del proyecto, sino como una fuente de inspiración para utilizar un proceso propio. Esto se debe, entre otras cosas, a que:

- SCRUM es un modelo de referencia para equipos de desarrollo.
- No tiene sentido unir los roles mínimos (*Scrum Master*, *Product Owner* y *Development Team*) en una única persona.
- No tiene sentido realizar *Daily Meetings*[4] un único desarrollador.
- No existe un cliente con el que revisar el trabajo completado tras cada Sprint.

2.1. Proceso de desarrollo

Se ha realizado, por tanto, un desarrollo basado en *Sprints* que añaden funcionalidad de forma iterativa e incremental a la aplicación.

En todos los *Sprint* se realizan diversas tareas en mayor o menor medida. Las tareas realizadas son las siguientes:

- **Análisis del alcance y la funcionalidad.** En todos los *Sprint* se ha decidido al inicio del mismo cuáles serían las funcionalidades principales a desarrollar.
- **Implementación.** Todas las fases del desarrollo han tenido una parte importante de implementación de funcionalidades.
- **Pruebas.** En todos los *Sprint* se han realizado pruebas para verificar el correcto funcionamiento de las funcionalidades implementadas. Conforme el proyecto ha avanzado estas pruebas han tenido un mayor peso, ya que hay más interacción entre distintas partes del sistema.
- **Documentación.** Cada funcionalidad conlleva la creación de nueva documentación que la apoye o la modificación de la que ya existía en fases previas.

La división prevista inicialmente en *Sprints* es la siguiente:

- Sprint 1. Implementación de los controles del vehículo.
- Sprint 2. Adición de circuito y colisiones.
- Sprint 3. Incorporación de contrincantes.
- Sprint 4. Creación de menús.

3 Cualquier aplicación que no se instale directamente desde Google Play Store es considerada de orígenes desconocidos.
4 Reuniones que realizan los equipos de desarrollo basados en SCRUM para compartir el estado actual de su trabajo, si han tenido problemas para hacerlo y qué planean hacer.

– Sprint 5. Modificaciones de vehículo.
– Sprint 6. Mejoras y pruebas globales.

Esta división está sujeta a cambios debidamente justificados.
Además, en un *Sprint* se puede modificar una funcionalidad implementada anteriormente si se encuentra algún error o si esto es necesario para desarrollar una funcionalidad nueva.
Todos los detalles sobre la distribución real de los *Sprints* se pueden encontrar en el apartado 3.2.2.

2.2. Tecnologías y herramientas

A continuación se listan las tecnologías y herramientas empleadas, así como su utilidad para el desarrollo del proyecto:

2.2.1. Cocos2d-x v3.2 (C++)

Tras estudiar otras posibilidades como Unity3D, AndEngine o LibGDX, este ha sido el *framework* seleccionado para desarrollar el proyecto.
Las razones principales son dos:

– Es una herramienta open-source y gratuita, tanto para proyectos comerciales como no comerciales.
– Posee una documentación bastante completa para iniciar el desarrollo, así como una comunidad muy activa y amplia.

Además, Cocos2d-x emplea C++ como lenguaje de desarrollo, siendo el más extendido en el desarrollo profesional de videojuegos y que ofrece un gran rendimiento en aplicaciones que precisen cargas de trabajo intensas de la CPU.

Por otra parte, Cocos2d-x tiene una capa muy reducida dependiente de la plataforma, por lo que permite exportar con pocas variaciones el proyecto a distintas plataformas (iOS, Android, Linux, Win32…).

2.2.2. Android NDK v.r9d

Android NDK es una herramienta que permite implementar partes de una aplicación Android en código nativo (como C++), a cambio de un aumento en la complejidad de dicha aplicación.
Lo que hace es que la ejecución de la aplicación sea directamente sobre el procesador y no interpretada por una máquina virtual.
Por lo tanto, si bien no es recomendable para cualquier aplicación, sí tiene cabida en aplicaciones que, como se ha comentado anteriormente, requieren una carga de trabajo importante de la CPU. Además de ser necesaria alguna herramienta que permita compilar código C++ en Android.

NOTA: En el momento en que se inicia el desarrollo, la versión r9d es la óptima para trabajar con Cocos2d-x, no siendo válidas versiones superiores.

2.2.3. Python v2.7.8

Python es un lenguaje de programación interpretado de alto nivel. Es un lenguaje muy sencillo que enfatiza en la legibilidad del código y el dinamismo, siendo muy utilizado para RAD[5] y como conector entre varios componentes o módulos escritos en C++ (como destacan en la web oficial *Comparing Python to other languages | Python.org*).
Es por esto que Cocos2d-x v3.0+ lo emplea para configurar y compilar los archivos de la aplicación.

NOTA: debido a ciertos cambios en la sintaxis de Python, Cocos2d-x requiere que se utilice Python v2.

2.2.4. Apache Ant v1.9.4

Apache Ant es una herramienta que se utiliza para la automatización de procesos de compilación y construcción.
Su función es similar a la de Make, pero no depende de las órdenes del Shell de cada SO (gracias a la Java Virtual Machine) y es más eficaz a la hora de determinar qué archivos están actualizados.

Es una herramienta ideal para soluciones multi-plataforma, al igual que Cocos2d-x.

2.2.5. Eclipse ADT Bundle

Éste paquete contiene el IDE Eclipse junto a Android SDK (ya configurado), por lo que es una opción muy recomendable para desarrollar aunque no imprescindible.

2.2.6. Adobe Photoshop CS6

Éste programa de edición de imagen 2D es imprescindible en un proyecto como éste, siendo utilizado para múltiples tareas, como recortar, escalar y orientar las imágenes que componen el juego.

2.2.7. Tiled v0.11

Tiled es una herramienta que permite crear escenarios para juegos 2D con orientación ortogonal, isométrica o hexagonal.
Tiled utiliza una imagen que contiene *tiles* (“baldosas”) de un tamaño uniforme, para crear un mapa apilando esos *tiles* y que permite configurar puntos de spawn[6], colisión…

La ventaja de utilizar un mapa en formato .tmx (similar a XML), además de lo ya mencionado, es que es muy sencillo abrirlo en el editor y sustituir un *tile* por otro en cualquier momento, para cambiar por ejemplo la disposición de los obstáculos u ornamentos.

[5] *Rapid Application Development*
[6] *El punto de spawn es como se conoce al lugar donde un personaje de un videojuego aparece por primera vez o reaparece después de morir.*

2.2.8. Visual Paradigm v11.2

Visual Paradigm es un programa para el diseño de Software que emplea UML[7]. Se ha utilizado para crear los diagramas de análisis y diseño del sistema (disponibles en el Manual Técnico).

3. Planificación y costes

En este apartado se detallan la planificación temporal prevista (*3.1*) y la adecuación real del proyecto a ésta (*3.2*).
También se detalla qué funcionalidades se han desarrollado en cada *Sprint* (*3.2.2*) y la justificación correspondiente a los desvíos producidos (*3.2.3*).

3.1. Planificación temporal

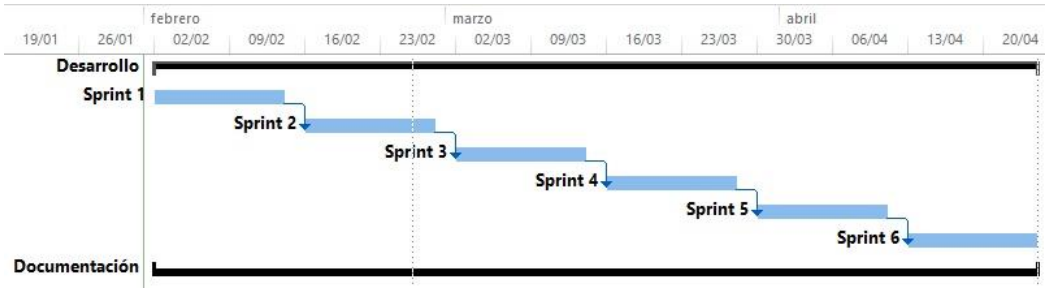
La planificación realizada inicialmente para la distribución de horas, teniendo en cuenta que se considera que el proyecto debe abarcar un total de 300 horas, es la siguiente:

Dedicación semanal prevista (en horas/semana): **25**	
Fase	**Estimación temporal** (en semanas)
Sprint 1	**2**
Sprint 2	**2**
Sprint 3	**2**
Sprint 4	**2**
Sprint 5	**2**
Sprint 6	**2**
TOTAL PROXECTO	**12**

A continuación se muestra el Diagrama de Gantt que representa la disposición de horas a lo largo del tiempo de desarrollo estimado.

	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1	**Desarrollo**	**12 sem.**	**lun 02/02/15**	**vie 24/04/15**	
2	Sprint 1	2 sem.	lun 02/02/15	vie 13/02/15	
3	Sprint 2	2 sem.	lun 16/02/15	vie 27/02/15	2
4	Sprint 3	2 sem.	lun 02/03/15	vie 13/03/15	3
5	Sprint 4	2 sem.	lun 16/03/15	vie 27/03/15	4
6	Sprint 5	2 sem.	lun 30/03/15	vie 10/04/15	5
7	Sprint 6	2 sem.	lun 13/04/15	vie 24/04/15	6
8	Documentación	12 sem.	lun 02/02/15	vie 24/04/15	

[7] *Unified Modeling Language*



Este método de desarrollo es susceptible a cambios durante el mismo.

3.2. Seguimiento

A pesar de la planificación, no ha sido posible mantener un ritmo constante de trabajo durante todo el periodo de desarrollo, por lo que se han producido desvíos en los plazos de finalización de cada sprint.

Éste ha sido el reparto real de horas del proyecto. Posteriormente se detallarán ciertos desvíos (*3.2.3*) causados por los problemas encontrados durante el desarrollo.

3.2.1. Reparto temporal real

Aquí se detalla en forma de tabla cuál ha sido el reparto de horas entre los distintos *Sprint*, así como la suma total de las horas reales que han sido necesarias para finalizar en tiempo y forma este proyecto.
Se ha decidido no crear un diagrama de Gantt (sí se hizo en la planificación) debido al reparto tan irregular de horas durante todo el proyecto.

FASE	TIEMPO (HORAS)
SPRINT 1	87,27
SPRINT 2	44,52
SPRINT 3	42,65
SPRINT 4	13,48
SPRINT 5	
SPRINT 6	
TOTAL	**187,92**

3.2.2. Funcionalidades por Sprint

A continuación se detallan las funcionalidades desarrolladas en cada Sprint, éstas han sido decididas al inicio de los mismos, en función de:

- Funcionalidades ya implementadas.
- Funcionalidades restantes.
- Tiempo de desarrollo real con respecto al estimado.

En la descripción de las funcionalidades desarrolladas en cada *Sprint* no se tienen en cuenta las tareas de documentación, ya que ésta va asociada en mayor o menor medida a cada una de las funcionalidades.

Nótese que se realizó *"Sprint 3. Creación de menús"* antes que *"Sprint 4. Incorporación de contrincantes"*, a pesar de que inicialmente estaba planeado que estos *Sprints* se realizaran en orden inverso (más detalles en 3.2.3.1).

3.2.2.1. Sprint 1. Implementación de los controles del vehículo

En este *Sprint* se implementa una primera versión de la escena de carrera, con un vehículo que se puede manejar a izquierda y derecha con dos flechas.

- Escena principal de carrera.
- Objeto jugador.
- Botones para controlar al jugador y eventos asociados a éstos.
- Mapa simple (estático).

3.2.2.2. Sprint 2. Adición de circuito y colisiones

En este *Sprint* se cambia el objeto mapa de una imagen de fondo estática a un objeto de tipo TMXTiled que se mueve hacia abajo y crea la ilusión de ser "infinito".
Se añaden obstáculos simples que se crean de forma aleatoria y el jugador puede colisionar con ellos.
También se añade una música de fondo y un efecto sonoro de colisión, así como las líneas de meta de cada vuelta y un texto que indica las vueltas superadas.

- Mapa TMXTiled, dinámico e infinito.
- Creación aleatoria de obstáculos y movimiento de los mismos.
- Control de colisiones entre jugador y obstáculos.
- Música de fondo y efectos sonoros básicos.
- Creación de líneas de vuelta y conteo de las mismas.

3.2.2.3. Sprint 3. Creación de menús

En este *Sprint* se diseñan y crean todos los menús del juego (escenas), con los botones/objetos correspondientes y toda la navegación entre escenas que se precisa para la versión final del juego.
Esto incluye el envío de información entre escenas, como pueden ser los parámetros de la carrera introducidos en el menú de configuración.
También se diseñan e introducen el logotipo del juego y el icono de la aplicación.

- Creación de todos los menús del juego (y botones correspondientes):
  - Menú principal.
  - Menú de configuración de carrera.
  - Menú de pausa durante la carrera.
  - Menú de fin de carrera.
- Adición de GUI (Graphic User Interface) durante la carrera.

3.2.2.4. Sprint 4. Incorporación de contrincantes

Este *Sprint* se centra en la adición de rivales a la carrera, así como crear un temporizador para guardar un registro del tiempo de cada vuelta.
Esto permitirá crear un *ranking* de vueltas rápidas asociadas al nombre introducido por el jugador en la escena de configuración de la carrera.

También se mejora el algoritmo de creación de obstáculos.

- Creación de rivales y lógica para que éstos eviten las colisiones con obstáculos.
- Creación de temporizador de vueltas.
- Mejora de creación de obstáculos.
- Creación de *ranking* de vueltas rápidas.

3.2.2.5. Sprint 5. Modificaciones de vehículo

3.2.2.6. Sprint 6. Mejoras y pruebas globales

3.2.3. Desvíos

En este apartado se detallan y justifican los desvíos producidos con respecto a la planificación inicial.

3.2.3.1. Cambio en el orden de *Sprints*

Durante el transcurso del proyecto, se decidió desarrollar *"Sprint 3. Creación de menús"* antes que *"Sprint 4. Incorporación de contrincantes"*.

Antes de iniciar este *Sprint*, se había desarrollado la mayor parte de la escena de carrera, y se consideró que sería interesante poder empezar a configurar la misma sin tener que modificar y recompilar el código fuente.
Además, la creación de los menús permitiría probar varias carreras sin la necesidad de cerrar la aplicación y abrirla de nuevo.

Éste cambio no implicó ningún desvío en el reparto de horas, ya que a pesar de la variación en el orden, ambos *Sprint* se desarrollaron.

3.2.3.2. Implementación de Box2D

El problema de *Implementación de Box2D* implicó cambios en el código, así como muchas horas de búsqueda de información y ayuda, lo que provocó un desvío aproximado de:

TAREA	DESVÍO (HORAS)
INVESTIGACIÓN	20
DESARROLLO	10
BÚSQUEDA DE RAZONES Y SOLUCIONES CORRESPONDIENTES	20
IMPLEMENTACIÓN DE SOLUCIÓN FINAL	10
TOTAL	**60,00**

3.3. Estimación de costes

4. Problemas, soluciones y conclusiones

A continuación se detallan los problemas y dificultades surgidos durante el desarrollo del proyecto, así como las soluciones adoptadas (de ser el caso).

4.1. Problemas encontrados

4.1.1. Implementación de Box2D

Inicialmente se decidió tratar de implementar el vehículo utilizando el motor de físicas Box2D (utilizado por Cocos2d-x), ya que parecía ser a priori la solución idónea, pero no se consiguió que dicho motor de físicas funcionase en el proyecto para obtener el resultado esperado.
A pesar de buscar una razón que lo justificase, no se ha podido confirmar ninguna de las posibles razones como causante del problema. Las posibilidades que se barajaron tras muchas horas de investigación son:

- **Tamaño de las imágenes mayor del soportado por el dispositivo.** Se emplearon las mismas imágenes en las versiones con y sin Box2D, por lo tanto:
  - **Descartado**
- **Errores en el código fuente.** Se encontró una versión de ejemplo similar a lo esperado en un repositorio público de la plataforma GitHub. Se probó el código fuente público y posteriormente, ya que no funcionaba, se contactó con el propietario del repositorio. Éste envió por email otra versión que sí debería funcionar, pero el resultado fue el mismo. Por lo tanto:
  - **Descartado**
- **Incompatibilidad del Hardware disponible.** Es posible que el recurso Hardware empleado para desarrollar la aplicación no soporte la inclusión de Box2D. Por ello se probó en otro modelo, en el que tampoco se podía arrancar la aplicación. Por lo tanto:
  - **No demostrable**

Finalmente, se decidió que el desvío comenzaba a ser elevado y habría que buscar una solución alternativa.

Solución: Simplificar las mecánicas del juego y descartar Box2D para el desarrollo.

4.1.2. Elevado coste de horas de optimización

Debido a la naturaleza jugable del proyecto, es necesario que el código sea lo más eficiente posible, para permitir que el número de *fps*[8] se mantenga constante.

[8] *Frames per second. Es la cantidad de imágenes estáticas por segundo que se muestran en la pantalla para generar sensación de movimiento*

El hecho de que haya muchas imágenes de forma simultánea en la pantalla, además de otras que no se muestran (como por ejemplo cuando un rival desaparece de la pantalla pero sigue moviéndose), ha dificultado esta tarea obligando a buscar siempre la mejor solución y no la más rápida o sencilla.
A la hora de buscar la mejor solución también se ha tenido en cuenta que el proyecto sería ampliable incluso después de finalizado.

Esto provoca que algunas funcionalidades hayan requerido un tiempo de desarrollo más elevado del que se podría esperar.

4.1.3. Dificultad para documentar

Este proyecto contiene poca interacción entre usuario y sistema, por lo que la documentación más habitual es insuficiente para ofrecer una vista muy detallada de todo el sistema.
Esto implica que la documentación debe reflejar las tareas internas más importantes y cómo se llevan a cabo, para lo que ha hecho falta buscar mucha información y ejemplos acerca del diseño Software para videojuegos.
La mayor parte de la información al respecto no es gratuita y esto ha supuesto un gran inconveniente.

4.2. Conclusiones

5. Mejoras y posibles ampliaciones

6. Referencias

- Video Games: 14 in the Collection, for Starters (Paola Antonelli, MoMA, 2012)
- Comparing Python to other languages | Python.org
- GitHub